Vorlesungsmitschrift

Semantik von Programmiersprachen

gelesen von Prof. Dr. Elfriede Fehr

Tobias Höppner

SoSe 2014

Inhaltsverzeichnis

1 Vorlesung 1 - 17.04.

1.1 Was sind Programmiersprachen?

Definition 1.1 (Programmiersprachen)**.**
Programmiersprachen sind künstliche, formale Ausdruckssprachen zur Kommunikation zwischen Mensch und Maschine.

Memo technischer Begriff -> z.b. ADD reg1 reg2

Beim Studium von Sprachen unterscheidet man 3 Ebenen(Aspekte):

Syntax einschließlich lexikalischer Struktur (Themen des Übersetzerbaus)
- Kern der Syntax ist die grammatikalische Struktur
- formale Definition durch kontextfreie Grammatiken

Semantik (diese Vorlesung)
- Bedeutung
- Interpretation
  Natürliche Sprachen (Gegenstand der Geisteswissenschaften) lassen Spielräume zur Interpretation offen. Künstliche Sprachen sollen möglichst formalisierbar sein.

Fokus: Formalisierung

Pragmatik Fragen nach dem Gebrauch und Zweck (Useability).
Warum sagt jemand xyz und ist das leicht verständlich?! - Was will jemand damit bewirken?)

1.2 Mehrdeutigkeit in natürlichen Sprachen

Synonyme *Schloss, Schimmel, ...*
Auflösung durch Kontext (meist leicht und unproblematisch)

Satzebene *Dieses Gelände wird zur Verhütung von Straftaten durch die Polizei Videoüberwacht.*
Auflösung durch Hintergrundwissen möglich. Weiteres Beispiel: *Staatsanwaltschaft ermittelt gegen Betrüger in Clownskostüm.*

1.3 Formalisierungsmethoden

In dieser Vorlesung werden drei Formalisierungsmethoden für die Semantik von Programmiersprachen behandelt.

Motivation

- Sicherheit beim Programmentwurf
- Formale Verifikation von Eigenschaften
- Richtlinie Übersetzerbau
- Automatische Erzeugung von Programm aus Spezifikation

Entwicklung der Formalisierungsansätze

operationale Semantik *(Landin 1964)*: Man stützt die Bedeutung auf die Funktionsweise technischer und abstrakte Maschinen ab. Dazu macht man die Maschine so einfach wie möglich und erkläre die Wirkung der Befehle auf die Maschine. Diese Semantik ist ähnlich ähnliche wie die denotationelle Semantik (mathematische Notation), jedoch wirklich näher an der Maschine.

denotationelle Semantik *(McCarthy 1962)*: Formales erfassen durch mathematische Notation. Weitgehende Abstraktion vom Zustandsraum mit einer direkten Zuordnung von syntaktischen Komponenten zu mathematischen Objekten (Semantik).

axiomatische Semantik *(Hoare 1969)*: Veränderung/Transformation von Bedingungen/Prädikaten auf dem Zustandsraum (einer abstrakten Maschine). Das geschieht mit mathematischen Formeln. z.B.: Hoareformel: $\{Q\}P\{R\}$

1.4 Referenzsprache

Um alle drei Formalisierungsmethoden zu betrachten nutzen wir die Referenzsprache **WHILE**.

1.4.1 Definition der Syntax

(Wie ist die Sprache grammatikalisch aufgebaut?!)
Elementare Einheiten

```
// ganze Zahlen (endlicher Ausschnitt der ganzen Zahlen MIN+1 .. MAX)
Z::= 0 |1|...|MAX|-1|-2|...|MIN
// Wahrheitswerte BOOL
W::= TRUE | FALSE
// Konstanten KON
K::= Z | W
// Bezeichner bzw. Variablen mit Indizes
I::= a|b|...|z|a1|a2|...|zi
// Operatoren
OP::= + | - | * | / | mod
// boolesche Operatoren
BOP::= < | > | = | !> | !< | !=
```

Zusätzliche Einheiten *(induktiv)*

```
// Terme TERM
T::= Z  |  I  |  T1 OP T2 | read, für T1,T2 in TERM
// boolesche Terme BT
B::= W | T1 BOP T2 | read | not B
// Befehle (Zustandstransformation) COM
C::= skip | I := T | C1; C | if B then C1 else C2 | while B do C |
    output T | output B
```

Die Indizes sind dazu da das Vorkommen von Symbolen in der Struktur *eindeutig* zu beschreiben.

Warum braucht man für so eine Sprache eine formale Semantik?!
Ich möchte maschinell arbeiten, aber es gibt Unterspezifikationen, unklar ist das Verhalten bei:

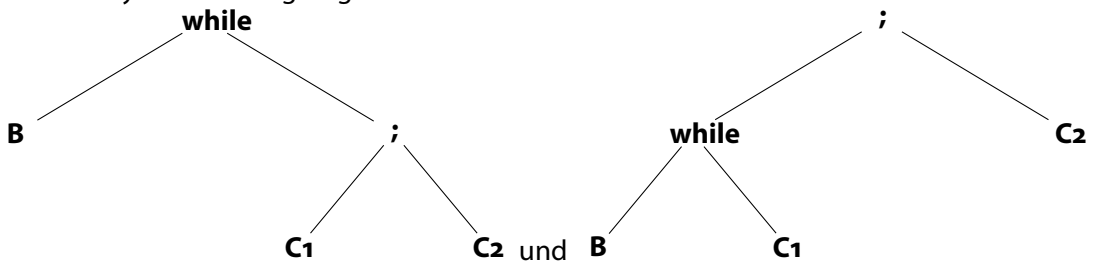
- Typkonflikte
- Fehlerbehandlung
- Rekursion

WHILE ist mehrdeutig?

Ja, das zeigt folgendes Beispiel:

```
while B do C1; C2
```

wo beide Syntaxbäume gültig sind.



Um dies zu verhindern werden untergeordnete Befehle eingerückt oder geklammert.

```
while B do
  C1;
  C2
```

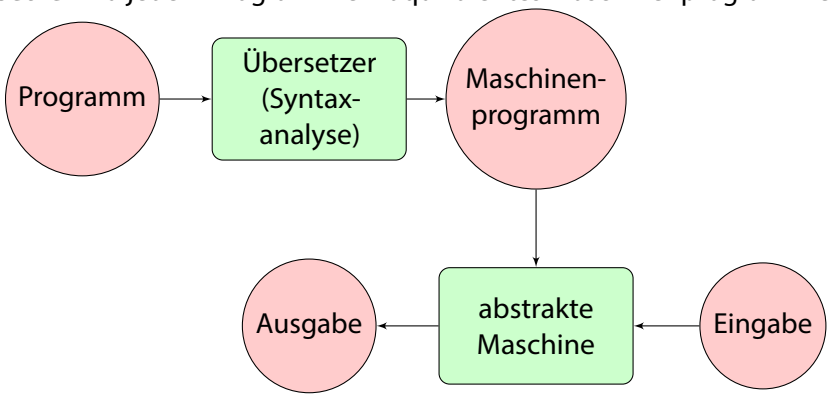
2 Vorlesung 2 - 24.04.

2.1 Operationelle Symantik am Beispiel der Terme

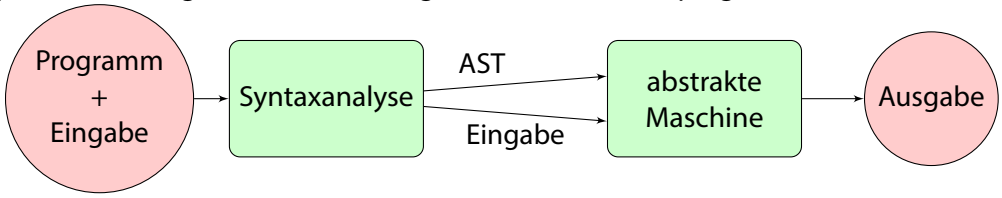
(Inhalt ist nicht im Lehrbuch!)

Es ist wichtig die Struktur von einer Sprache zu kennen, erst dann kann man eine korrekte Interpretation anfertigen!
Grundsätzlich gibt es zwei Methoden:

Übersetzer Zu jedem Programm ein äquivalentes Maschinenprogramm erstellen.



Interpreter Das Programm wird mit Eingabe zum Maschinenprogramm transferiert.



AST: abstract syntax tree

2.1.1 Terme der Sprache WHILE

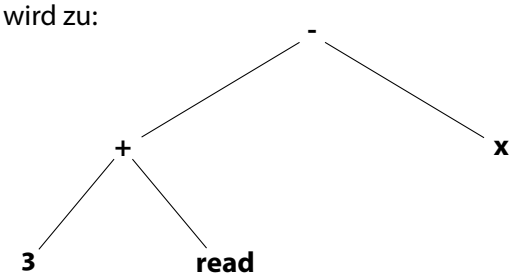
```
// Terme TERM
T::= Z  |  I  |  T1 OP T2 | read, für T1,T2 in TERM
```

2.1.2 Beispiel: AST

Der Ausdruck

```
3 + read - x
```

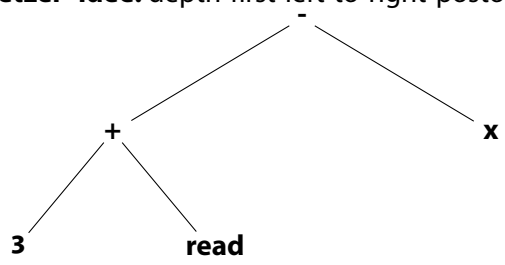
wird zu:



2.1.3 Informelle Semantik

Interpretation nur möglich, wenn Speicher und Eingabe vorgelegt sind.
Annahme: wir bekommen alles als AST und wir bekommen eine Eingabe, die auch von der Maschine unterstützt wird!

Übersetzer **Idee:** depth-first-left-to-right-postorder Traversierung des AST Unser Beispiel:



wird übersetzt zu:

```
1 PUSH 3
2 READ
3 ADD
4 LOAD x
5 SUB
```

Zustandsveränderungen:
aktueller Zustand (siehe Bild Architektur!): <e|S|8.5. ...> -PUSH 3> <3.|S|8.5. ...> -READ> <-8.3-e|S|5. ...> -ADD> <3+(-8).e|S|5. ...> -LOAD x> <2.-5.e|S| ...5> -SUB> <-7.e|S|5...>
Semantik eines Terms T zu geg. Speicher S und Eingabe E ist die Spitze des Wertekellers(STACK) nach Ausführung von trans T auf <e|S|E>, falls diese Ausführung fehlerfrei läuft, sonst Fehler!

Interpreter (abstrakte Maschine beinhaltet eine Komponente (Kontrollkeller), in der ASTs in einem Keller gespeichert werden können.)

- Kontrollkeller
- Zustand der abstrakten Maschine hat Komponenten
  * Wertekeller W ($\in ZAHL*$)
  * Speicher S ($S \in [ID \to Zahl]$)
  * Kontrollkeller K ($\in (AST \cup OP)*$)
  * Eingabe E ($\in ZAHL*$)

Zur Formalisierung der Semantik über die abstrakte Maschine mit dem Zustandsraum Z durch Angabe von:

(i) einem Anfangszustand $Z_{T,S,E}$ für jeden Term T, Speicher S und Eingabe E.
(ii) eine Zustandsüberführungsfunktion $\Delta : Z \to Z$ (partiell)
(iii) Erklärung der Semantik über Iteration von Δ
für Terme aus WHILE:

(i) $Z_{T_0,S_0,E_0} :=< e|S_0|T_0.e|E_0 >$
(ii) Δ per Induktion über die Struktur der Kontrollkellerspitze
$\Delta < W|S|n.K|E >=< n.W|S|K|E >$ für alle $n \in ZAHL, W, S, E$ wie oben.
$\Delta < W|S|x.K|E >=< s(x).W|S|K|E >$ für alle $x \in ID$
$\Delta < W|S|read.K|n.E >=< n.W|S|K|E >$ für alle $n \in ZAHL$
$\Delta < W|S|T_1OPT_2.K|E >=< W|S|T_1.T_2.OP.K|E >$
$\Delta < n_2.n_1.W|S|OP.K|E >=< n_1OPn_2.W|S|K|E > n_1, n_2 \in ZAHL$ falls n_1OPn_2 definiert ist.

(iii) Die Semantik eines (beliebigen) Terms T im Bezug auf einem Speicher S und eine Eingabe E ist $n \in ZAHL$, wenn $\Delta^k Z_{T,S,E} =< n.e|S|e|E' >$ für beliebige $E' \in Zahl*$, undefiniert sonst!

Architektur der abst. Maschine

Befehlssatz

```
// Stack und Speicher Operationen
READ    // nimm Zahl von Eingabe, lege auf Stack, rücke Zeiger um eins
    weiter
LOAD x    // nimm Inhalt aus Speicher mit symbolischer Adresse x und lege
    auf Stack
PUSH n    // für jede Zahl aus N lege n auf Stack
(STORE x) // belegt Speicher mit der symbolischer Adresse x
(GOTO n)  // bedingter Sprung
// arithmetische Operationen
ADD
MULT
SUB
```